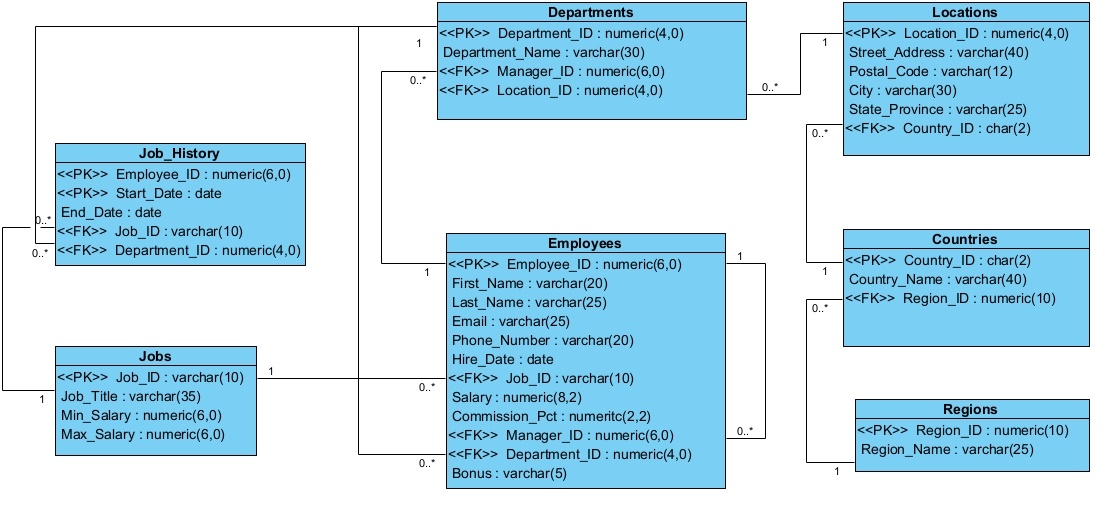
Databases 6G4Z0016 Labsheet

# Topic 3 – SQL Joins



# Worked Examples

### Q1. List the names of the employees and the names of the departments they work in.

If we examine the ERD above we see that the employee names are in one table (Employees) but the department names are in a different table (Departments), we are therefore being asked to get information from two different tables and present them as if they were in one table. This is what joins are for.

We start by identifying the columns we need and which tables they are in – first\_name, last\_name in the Employees table and department\_name from the Departments table.

Next, we have to identify how to join the tables. In this case, we know we need an inner join because for every row in the Employees table we want to find a single row in the Departments table that matches it. We need to provide the condition for the match – i.e. how will the database know when a row from the Departments table matches the row from the Employees table. Looking at the ERD again we see that both have a column called department\_id. The join condition is that the row in the Departments table has the same value for department\_id as the row in the Employees table.

We can therefore write the required SQL query:

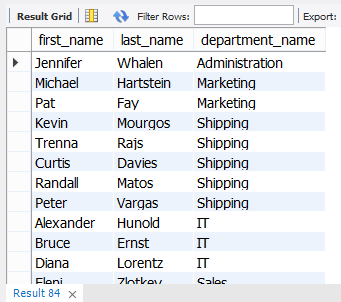
SELECT first\_name, last\_name, department\_name

FROM Employees

JOIN Departments

ON Employees.department\_id = Departments.department\_id;

This gives the following result:



Notice that we only have 19 records here whereas “SELECT \* FROM Employees;” shows 20 records. This is because the record for Kimberley Grant has NULL for the department\_id which cannot match anything in the Departments table and so no record is returned for her.

**IMPORT NOTE**

We say that we are “joining on” the department\_id column because that is the column used to match rows. It might be useful, therefore, to include that in the output. Try the following SQL:

SELECT first\_name, last\_name, department\_name, department\_id

FROM Employees

JOIN Departments

ON Employees.department\_id = Departments.department\_id;

This throws an error: “column … in field list is ambiguous”. The error means that one of the columns can have more than one meaning – i.e. the database cannot determine which column you are actually referring to. This is because there are two columns called department\_id involved in the query and we have not specified which one we mean.

The solution is to specify not only the column name but also the table name:

SELECT first\_name, last\_name, department\_name,

E**mployees.department\_id**

FROM Employees

JOIN Departments

ON Employees.department\_id = Departments.department\_id;

Notice that the error will be thrown if the table name is missed in the ON clause as well.

Try the following SQL:

SELECT first\_name, last\_name, department\_name,

Employees.department\_id

FROM Employees

JOIN Departments

**ON department\_id = department\_id;**

You will again get the same error.

**VERY IMPORT NOTE**

We mentioned in the lecture about the dangers of the Natural Join. This question is a good example. We might be tempted to use the Natural Join here.

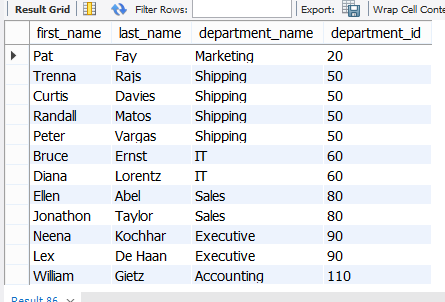
Try the following SQL:

SELECT first\_name, last\_name, department\_name, department\_id

FROM Employees

NATURAL JOIN Departments;

The results should be:



This is different to the result we got when using the more fully-specified INNER JOIN … ON method. The reason is that the Employees and Departments tables don’t just share a column called department\_id. They also both have a column called manager\_id. Therefore, the Natural Join will join on both of these columns. The equivalent SQL for the Natural Join of these two tables is:

SELECT first\_name, last\_name, department\_name,

Employees.department\_id

FROM Employees

JOIN Departments

ON Employees.department\_id = Departments.department\_id

**AND Employees.manager\_id = Departments.manager\_id;**

Notice the additional join condition which is what causes the results to be different. This mistake can happen if you use a natural join but is impossible using the INNER JOIN … ON method. Even if you were very careful when using the natural join, it is easy to imagine a situation in which the Departments table did not originally have a manager\_id in it and so the two queries were identical. Then, at some later date, the manager\_id column was added to the Departments table and now, without changing the SQL of the query at all, the results change and we are returning the wrong information.

This is why you should be very careful with and avoid using the natural join.

### Q2. List the last name, job title and department name for all employees who work in Toronto.

Looking at the ERD we can see that the columns we need for this query are all in different tables. last\_name is in the Employees table, job\_title is in the Jobs table, department\_name in the Departments table and the city in the Locations table. We will therefore need to join all these table together.

The join type will be inner joins because we want to find the rows that match so we find the right job title for the employee and the city for their department etc. Finally, because the question asks us to limit out results to the employees working in Toronto, we know we will have a WHERE clause to filter the results at the end.

Putting it all together we get the following SQL:

SELECT last\_name, job\_title, e.department\_id, department\_name

FROM Employees AS e

INNER JOIN Departments AS d

ON e.department\_id = d.department\_id

INNER JOIN Jobs AS j

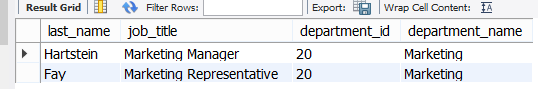
ON e.job\_id = j.job\_id

INNER JOIN Locations AS l

ON d.location\_id = l.location\_id

WHERE l.city = ‘Toronto’;

The results should be:



Notice that we have used aliases for the table names. This is because the join conditions always use columns with the same name in both tables and so we must specify the table name in order to avoid ambiguity. To make the query more readable we can use sensible aliases so that we can then write e.job\_id rather than the full Employees.job\_id.

An alternative way to write the same query would be to make use of the USING keyword:

SELECT last\_name, job\_title, department\_id, department\_name

FROM Employees

INNER JOIN Departments USING(department\_id)

INNER JOIN Jobs USING (job\_id)

INNER JOIN Locations USING (location\_id)

WHERE city = 'Toronto';

Personally, I think this is neater than using ON and is just as clear. However, it is rarer to see USING than ON because it can only be used when the join columns have the same name whereas ON can always be used. Some may argue also that ON is more verbose and therefore clearer and ON is supported by all relational databases whereas some may not support USING (although all the major ones do).

### Q3. Show all employees (their employee IDs and names) and their managers (their manager IDs and names). In both cases, combine first names and last names into one field.

The tricky bit of this question is that the employees and the managers are both in the same table but on different rows. We know by now how to combine rows from different tables but how can we combine rows in the same table?

The answer is to pretend that there are two tables called Employees and to join them together. When we write a query that joins a table with itself it is called a **self-join**. There is only one version of the table, obviously, but conceptually we can think of it as if there were two copies. We will have to use aliases for the table names because otherwise we will have two tables with the same name and all the columns will have the same names so everything will be ambiguous.

Once we understand that this is a self-join then the SQL is fairly standard:

SELECT e.employee\_id AS ‘Employee ID’,   
 CONCAT(e.first\_name, ' ' , e.last\_name) AS ‘Employee Name’,   
 m.employee\_ID AS ‘Manager ID’,   
 CONCAT(m.first\_name, ' ' , m.last\_name) AS ‘Manager Name’  
 FROM Employees AS e   
 INNER JOIN Employees AS m  
 ON e.manager\_id = m.employee\_id;

The results should be:



Notice that we use e and m as table aliases to stand for employee and manager. This means that when we do the join condition it is obvious that we need the manager ID from e and the employee ID from m. If we had chosen less clear aliases, say e1 and e2, then that would have been harder to figure out. If we had put them the wrong way around then we would end up with a syntactically correct query but one that was semantically wrong and gave the wrong results.

### Q4. Management is considering a major restructuring. They are considering creating local versions of every department in every country the company operates in. List the new departments that would have to be created in each country.

This question is quite complex so it’s helpful to think about what the output should look like. If we run “SELECT \* FROM Departments;” we will see that currently there is only a single version of each department in a single location. If we ran a query which joined Departments, Locations and Countries we could list the departments and the names of the countries they are located in. The SQL would be:

SELECT department\_name, country\_name

FROM Departments

NATURAL JOIN Locations

NATURAL JOIN Countries;

We can use the natural join because this query is only for our own understanding, not for production code.

The result of this query is:



The question is asking for a list of new departments to be created in each country. If we run “SELECT country\_name FROM Countries;” we will get a list of the four countries that the company operates in (USA, UK, Canada and Germany). So, if the company currently has an Administration department in the USA, the question is asking us to list that it needs to create an Administration department in Canada, Germany and the UK. What we are being asked to do is combine every possible combination of department\_name and country\_name and then remove those combinations that already exist in the database.

First, let us consider how to generate every possible combination. In fact, this is a straightforward application of the Cross Join:

SELECT department\_name, country\_name

FROM Departments

CROSS JOIN Countries;

Next, how will we remove the combinations that already exist? Since we are talking about removing rows from the result we need a WHERE clause. We want to remove any row where the combination of department\_name and country\_name already appears. Unfortunately, there is no common column between departments and countries and so we cannot usefully perform an inner join between these two tables to find out which countries each department is in at the moment. We will have to join the Locations table as well. This will allow us to list, for each department in the Departments table, its country\_id which we can get from the Locations table. If we do this, then we can use the WHERE clause to remove rows where the country\_id of the department matches the country\_id of the country we are generating a new pair with via the Cross Join.

But we don’t want to actually show the location\_id or country\_id. Fortunately, we can include a join without actually displaying any columns from the joined table. To make the results more readable we can also order them alphabetically by the name of the department. Our SQL is therefore:

SELECT department\_name, country\_name

FROM Departments

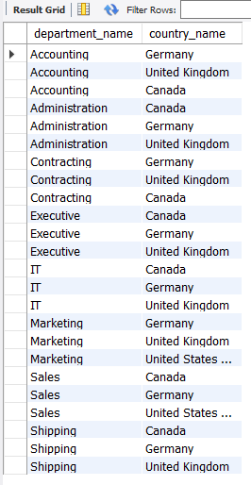
CROSS JOIN Countries

INNER JOIN Locations USING (location\_id)

WHERE Locations.country\_id != Countries.country\_id

ORDER BY department\_name;

The results should be:



### Q5. Show all employees (their employee IDs and names) and their managers (their manager IDs and names). In both cases, combine first names and last names into one field. If an employee has no manager, still include the employee.

This is essentially the same as Q3 above but this time we have to include all employees, even those without managers. Recall our answer to Q3:

SELECT e.employee\_id AS ‘Employee ID’,   
 CONCAT(e.first\_name, ' ' , e.last\_name) AS ‘Employee Name’,   
 m.employee\_ID AS ‘Manager ID’,   
 CONCAT(m.first\_name, ' ' , m.last\_name) AS ‘Manager Name’  
 FROM Employees AS e   
 INNER JOIN Employees AS m  
 ON e.manager\_id = m.employee\_id;

For that question, we used an inner join to perform the self-join. The inner join will only return rows that match based on the join condition. But if an employee has no manager then the manager\_id on their row will be NULL and NULL won’t match, so that row will not be returned at all.

The solution is to use the left join which will return the row from the left table even if there is no match in the right table. Recall that in SQL the left table is the one we list first. So all we need to do is use the same SQL as for Q3 but change INNER to LEFT OUTER:

SELECT e.employee\_id AS ‘Employee ID’,   
 CONCAT(e.first\_name, ' ' , e.last\_name) AS ‘Employee Name’,   
 m.employee\_ID AS ‘Manager ID’,   
 CONCAT(m.first\_name, ' ' , m.last\_name) AS ‘Manager Name’  
 FROM Employees AS e   
 **LEFT OUTER** JOIN Employees AS m  
 ON e.manager\_id = m.employee\_id;

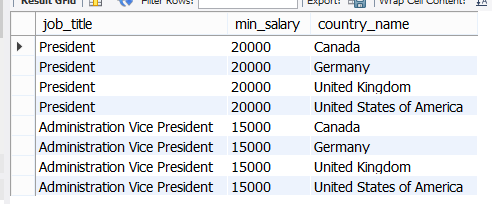
The result is now:



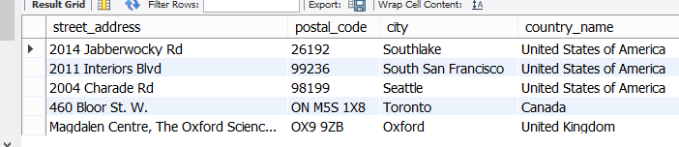
# Completion Problems

### Q1. List the job title and minimum salary for every job where the minimum salary is more than 10,000. Combine the output with every country the company operates in.

SELECT job\_title, min\_salary, country\_name   
FROM Jobs   
WHERE min\_salary > 10000;

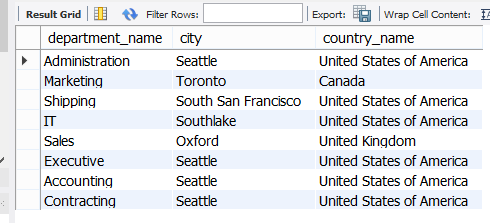


### Q2. List the full address of every location, including the country name.

SELECT street\_address, postal\_code, city, country\_name   
FROM Locations   


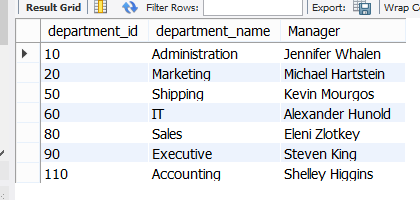
### Q3. List the name, city and country of every department.

SELECT department\_name, city, country\_name   
FROM Departments   
INNER JOIN Countries   
 USING (country\_id);



### Q4. List the department ID, name and manager name for every department.

SELECT d.department\_id, department\_name,   
 CONCAT(first\_name, ' ', last\_name) AS Manager  
FROM Departments AS d   
INNER JOIN Employees AS e



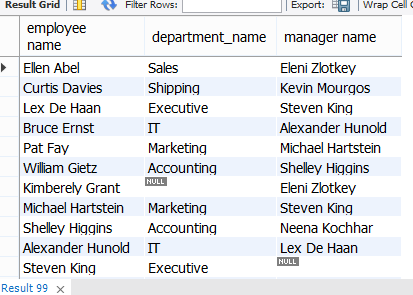
### Q5. List every pair of employees working in the same department, but only list each pair once. Order the results by department ID then by the surname of the first employee and then the surname of the second employee.

SELECT e.department\_id,   
 CONCAT(e.first\_name, ' ', e.last\_name) AS Employee,   
 CONCAT(c.first\_name, ' ', c.last\_name) AS Colleague   
FROM Employees AS e   
INNER JOIN Employees AS c   
ORDER BY e.department\_id, e.last\_name, c.last\_name;



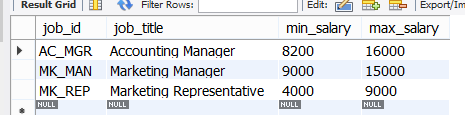
### Q6. List every employee, their department name and manager name. If they have no manager or department, still include them.

SELECT CONCAT(e.first\_name, ' ', e.last\_name) AS ‘employee name’,   
 department\_name,   
 CONCAT(m.first\_name, ' ', m.last\_name) AS ‘manager name’   
FROM Employees AS e   
ORDER BY e.last\_name;



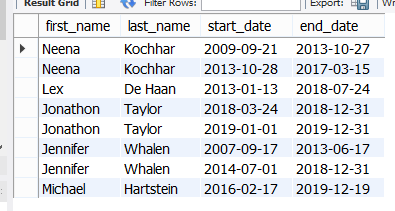
### Q7. List the details for all jobs with the letters “ing” in the title.

SELECT \*   
FROM Jobs



### Q8. List the information about employees who have left the company (i.e. have an end date).

SELECT first\_name, last\_name, start\_date, end\_date   
INNER JOIN Job\_History   
 USING (employee\_id);

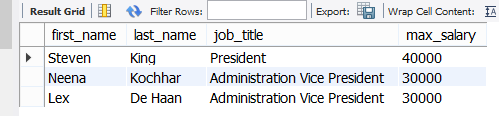


# Deliberate Practice: Write the SQL

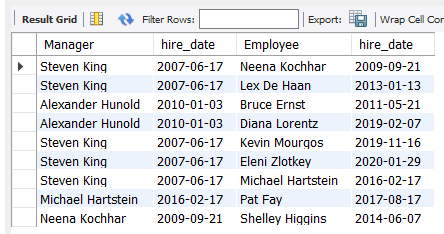
### Q1. List the names of all employees and their job titles, ordered alphabetically by surname.



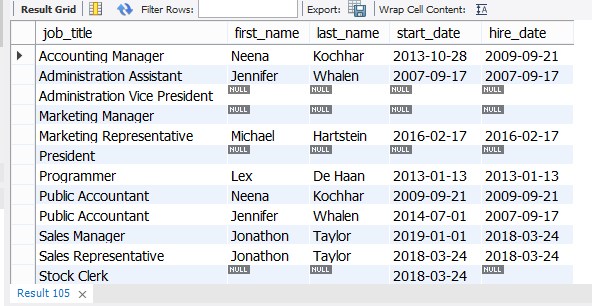
### Q2. List the names of the three employees with the highest potential salary and their job title



### Q3. The HR department wants a list of all employees who were hired after their managers. Produce a query that shows the names and hire dates of managers and the employees who they manage. Only show employees who were hired after their managers.

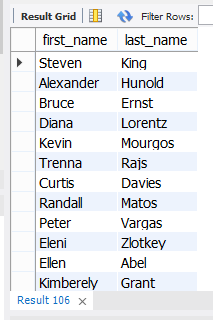


### Q4. List the job titles and names of employees who used to hold those positions, including their start and end dates. Where nobody has ever held that position only list the job titles and where the position is still being filled list the job title, employee name and start date. Give the results in order of job title.

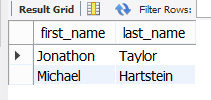


### Q5. List the names of all employees still working for the company in the same role they started with.

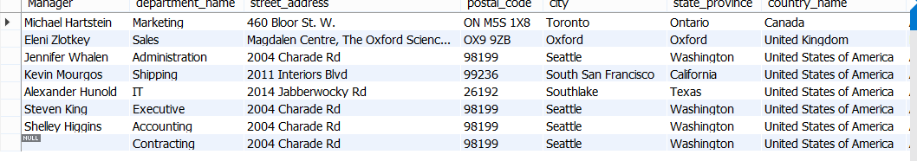
Note that the Employees table lists current roles and the Job\_History table lists any previous roles (but does not include current roles).



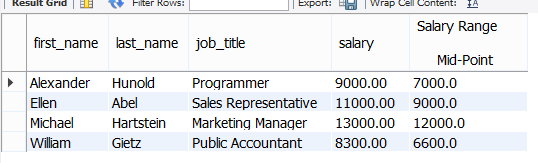
### Q6. List all employees who changed role in 2019.



### Q7. List the manager name, department name and full address including country and region name for every department – including those with no manager.



### Q8. List the employees who are in the upper half of their job’s salary range, include their job title, their salary and the mid-point of their job’s salary range.



### Q9. List the names of all employees and the countries where they work.

